

# 《React 思維進化》 ch2-8~2-9

畫面更新的發動機 : state & 畫面更新的流程機制 :  
reconciliation

Speaker : Monica

2024.04.03 @Tech-Book-Community

# 什麼是 state?

- 前端常遇到使用者與網頁互動，進而使網頁產生變化的情境
  - 需記錄這些「可更新的資料」以維持應用運作，在資料更新時連動更新畫面，此類資料稱為應用程式的「state(狀態資料)」
- 單向資料流：原始資料更新時，畫面才會更新，原始資料是畫面結果的起點
  - React 的 state 機制扮演「可更新的原始資料（也常稱作狀態）」角色，作為單向資料流起



# state 與 component

- React 採一律重繪策略，但只重繪跟被更新資料有關的畫面區塊

怎麼知道哪些畫面跟被更新的資料有關？

以 component 作為 state 運作的載體及一律重繪的界線

- 運作的載體
  - state 需依附於 component 才能記憶、維持狀態資料，生命週期隨 component 存亡
  - 可將 state 視為「component 內的資料記憶體」
- 一律重繪的界線
  - state 更新並啟動重繪時，只重繪該 component ( 包含其子孫 component ) 以內的畫面區塊

# useState 初探

- `useState` hook 可定義、更新狀態資料，並觸發 component re-render，進而更新瀏覽器畫面
  - 只能在 component function 內呼叫
  - 可想成是一種「在 component 內註冊並存取狀態資料」的工具

 Hooks : React 提供的 API，只能在 function component 內的頂層作用域才能呼叫的特殊函式，可將 React 核心特性或功能注入到 component 中

# useState 使用方式

```
1 import { useState } from 'react';
2 export default function App(props) {
3     const [state, setState] = useState(initialState);
4     // ...
5 }
```

- 參數：state 初始值，可以是任意型別的值
- 回傳值：一個陣列，陣列包含兩個項目
  - 第一個項目：「該次 render 的當前 state 值」
  - 第二個項目：「用來更新 state 值的 `setState` 方法」，是一個 JavaScript 函式
    - 呼叫 `setState` 時傳入新 state 值作為參數，以取代舊 state 值，並觸發 component re-render
- 開發慣例
  - 以陣列解構取得 state 值和 `setState` 方法
  - 根據商業邏輯自訂變數名稱，如：`const [count, setCount] = useState(0);`

# useState 應用範例

```
3  //呼叫 useState 定義一個 state，來記憶計數器的值，且初始值為0
4  const [count, setCount] = useState(0);
5
6  const handleDecrementButtonClick = () => {
7      //以參數指定新的 state 值為目前 count-1
8      setCount(count - 1);
9  };
10
11 const handleIncrementButtonClick = () => {
12     //以參數指定新的 state 值為目前 count+1
13     setCount(count + 1);
14 };
15
16 return (
17     <div>
18         <button onClick={handleDecrementButtonClick}>-</button>
19         {/* count 是 useState 取出的 state 值，count 一開始會是我們給的初始值 0 */}
20         <span>{count}</span>
21         <button onClick={handleIncrementButtonClick}>+</button>
22     </div>
23 );
```

# onClick 事件綁定

- 在 React，可透過 React element 來間接管理和綁定事件到實際 DOM element 上

## 如何綁定事件？

- 在對應實際 DOM element 類型的 React element 上新增 onClick prop，並傳遞事件處理函式給該 prop

```
1  function MyButton() {  
2      const handleClick = () => {  
3          console.log('click!');  
4      };  
5      return <button onClick={handleClick}>click me</button>;  
6  }
```

# 補充：只有對應實際 DOM element 的 React element 才會內建事件綁定的 prop

React element 可分為 3 種類型：

- 可對應實際 DOM element，如：`<h1>`、`<p>`、`<button>`
  - 只有此類才會內建事件綁定的 prop
  - 例如：`<button onClick={handleClick}>`，React 會自動綁定事件到該 button，類似幫你做`button.addEventListener("click", handleClick);`
- 對應自定義的 component function，如：`<MyComponent>`
  - 傳遞事件綁定的 props 沒有任何效果，只是剛好自定義 prop 也叫 `onClick`
- Fragment 類型，也就是 `<>` 或 `<Fragment>`
  - 傳遞 `key` 以外的其他 props 沒有意義

# setState 方法

- 點擊 + 按鈕，呼叫 `setState` 觸發狀態資料更新，並連動畫面更新
- `setState` 使用方式
  - 參數：要更新的新值，可以是任何型別的值
    - 如果傳入函式，此函式會被視為 updater function，updater function 會拿到一個 pending state 作為參數，並回傳要更新的新值（[官網範例](#)）
  - 回傳值：無
- `setState` 觸發 component 的 re-render 時，會重新執行 component function，產生新版本的 React element
  - 再次執行 `useState`，得到的回傳值 state 就是新 state 值（上次 `setState` 傳入的新值）

 呼叫 `setState` 後，React 不會立即觸發 re-render，而是等正在執行的事件內所有程式結束後，才開始執行 re-render，因此會聽到「`setState` 是非同步的」這種說法

# 使用 setState 方法

Counter component function 首次 render

Counter.jsx ×

src > Counter.jsx > ...

```
1 import { useState } from "react";
2 export default function Counter() {
3   const [count, setCount] = useState(0); → count 的值是呼叫 useState 傳入的預設值 0
4
5   const handleDecrementButtonClick = () => {
6     setCount(count - 1); → 所謂 count-1 其實就是 0-1，也就是 setCount(-1)，指定這個 state 的值要更新為 -1
7   };
8
9   const handleIncrementButtonClick = () => {
10    setCount(count + 1); → 所謂 count+1 其實就是 0+1，也就是 setCount(1)，指定這個 state 的值要更新為 1
11  };
12
13  return (
14    <div>
15      <button onClick={handleDecrementButtonClick}>-</button>
16      <span>{count}</span>
17      <button onClick={handleIncrementButtonClick}>+</button>
18    </div>
19  );
20}
```

# 使用 setState 方法

點擊 + 觸發 handleIncrementButtonClick 事件，  
呼叫 setCount(1)，觸發 Counter component function 的 re-render

```
Counter.jsx X
src > Counter.jsx > ...
1 import { useState } from "react";
2 export default function Counter() {
3   const [count, setCount] = useState(0); → count 的值是剛剛 setCount 指定要更新的值，也就是 1
4
5   const handleDecrementButtonClick = () => {
6     setCount(count - 1); → 所謂 count-1 其實就是 1-1，也就是 setCount(0)，指定這個 state 的值要更新為 0
7   };
8
9   const handleIncrementButtonClick = () => {
10    setCount(count + 1); → 所謂 count+1 其實就是 1+1，也就是 setCount(2)，指定這個 state 的值要更新為 2
11  };
12
13  return (
14    <div>
15      <button onClick={handleDecrementButtonClick}>-</button>
16      <span>{count}</span>
17      <button onClick={handleIncrementButtonClick}>+</button>
18    </div>
19  );
20}
```

# state 的補充觀念

## Hooks 的限制

- 只能在 component function 內被呼叫，hooks 需依賴 component 才能運作
- 只能在 component function 的頂層作用域被呼叫，不能在條件式、迴圈或 callback 函式中呼叫

```
1  function MyComponent() {  
2      // ✅ 合法的 hooks 呼叫：在 component function 的頂層作用域呼叫  
3      useState();  
4  
5      if( ... ){  
6          // ❌ 非法的 hooks 呼叫，沒有在 component function 的頂層作用域呼叫  
7          useState();  
8      }  
9      for( ... ){  
10         // ❌ 非法的 hooks 呼叫，沒有在 component function 的頂層作用域呼叫
```

## 為何 hooks 有這些限制？

- 為了確保 hooks 機制正確運作，沒遵守可能導致資料丟失問題

# state 的補充觀念

## 為什麼 useState 的回傳值是一個陣列

- useState 回傳值：[該次 render 的當前狀態值，更新狀態值的 setState 方法]
  - 回傳值是陣列有助於：呼叫 useState 後，更方便的將回傳值解構賦值給自定義變數
- 如果回傳的是其他資料型別?
  - useState 會回傳兩個值，需要用陣列或物件這種集合的方式來回傳
  - 如果回傳的是物件?
    - 每次都要解構回傳值、並為物件屬性賦予別名
    - 語法上不簡潔

```
1 //➑ 假想回傳物件的情況，非真實 useState 用法
2 const { state, setState } = useState();
3
4 //物件解構時為屬性賦予別名
5 const { state: count, setState: setCount } = useState(0);
6 const { state: isOpen, setState: setIsOpen } = useState(false);
```

# state 的補充觀念

## setState 方法是更新 state 值並觸發 re-render 的唯一合法手段

- 如何更新 state 資料的值?
  - setState 是唯一的更新方式
  - state 是單向資料流起點，資料變更後，才會驅動 React re-render，再更新對應的畫面區塊
- 如果不透過 setState 方法，自己修改 state 值會怎樣?
  - React 不知道你修改 state 的值，因此不會觸發 React 的 re-render，也無法讓對應的畫面更新
    - 導致資料與畫面不同步，單向資料流可靠性被破壞
  - 書中範例：直接修改 state 資料無法觸發 re-render

# state 的補充觀念

## React 如何辨認同一個 component 中的多個 state

- 可在 component function 內多次呼叫 `useState` 來定義不同的 state，且每個 state 值互不影響：

```
const [count, setCount] = useState(0);
const [name, setName] = useState('Chair');
```

- 沒有給 id 或 key 這類的值，React 怎麼知道哪次的 `useState` 要回傳哪個 state 的資料？
  - component 的所有 hooks 在每次 render 都會依賴固定的呼叫順序，以區別彼此
    - 對應 hooks 的限制：hooks 只能在 component function 頂層作用域被呼叫
    - 在頂層作用域呼叫 hooks，才能保證每次 render 時，hooks 都會被呼叫、且執行順序固定不變
- 多次呼叫 hooks 時，React 記的是「第一個呼叫的 hook」、「第二個呼叫的 hook」這種順序，以順序區別

# state 的補充觀念

同一個 component 的同一個 state，在該 component 的不同實例間的狀態資料獨立

- component 是一種藍圖，可透過藍圖產出實例，產出的實例互不影響
  - state 是依附在 component 上的資料，透過 component 藍圖產出的實例所擁有的 state 也互不影響
  - React component state demo

```
1 import Counter from './Counter';
2 export default function App() {
3     //這三個 Counter component 實例的 counter state 資料是獨立、互不影響的
4     //第一個 Counter component 的 count 的值變動，不會影響另一個 Counter component 的 count 值
5     return (
6         <div className='App'>
7             <Counter />
8             <Counter />
9             <Counter />
10        </div>
11    );
12 }
```

# 補充：props 與 state 的差異

		state	props	
同		都是與資料相關的機制		
異	目的	維護狀態資料，作為「更新資料後，一律重繪以連動更新畫面」的啟動點	傳遞資料給子 component	
	核心理念	「儲存資料」的手段，儲存應用程式狀態資料的載體	「傳遞資料」的手段，父層 component 傳遞資料給子層 component 的管道	
	什麼時候需要用	需透過更新資料來連動影響畫面時，這些資料都應以 state 管理，以確保單向資料流正常運作	要從父 component 往下傳遞資料給子 component 時	
	補充	如果是純靜態應用，則不須使用 state	為維持單向資料流一致性，不應直接修改 props	

| props 與 state 本質不同，使用情境也不同，不存在二選一的問題

# Render phase 與 Commit phase

React 的畫面處理機制可分為兩階段：

- ① 產生一份描述最新畫面結構的 React element
  - 對應 component 的處理機制，稱為「render phase」
  - 在 render phase，component 會渲染並產生 React element
- ② 將 React element 轉換為畫面上實際的 DOM element
  - 對應 component 的處理機制，稱為「commit phase」
  - 在 commit phase，component 會將 React element 提交並處理到瀏覽器的實際 DOM element

# Render phase 與 Commit phase

產生初始畫面時：首次 render component

## Render phase

- 執行 component function，以 props 與 state 資料產生初始畫面的 React element
- 將產出的 React element 交給 commit phase 處理

## Commit phase

- 將 component 在 render phase 回傳的 React element 全部轉換、建立成實際 DOM element
    - (因為第一次 render 時，瀏覽器畫面還沒有此 component 對應的實際 DOM element)
  - 透過瀏覽器 API `appendChild()` 放到實際畫面上
- 「component 首次 render 並 commit 到實際 DOM」的過程也稱為「mount」
  - mount 完成的狀態稱為「mounted」，代表 component render 已完成，且已「掛載」到瀏覽器畫面
    - mounted 後，才能在瀏覽器結構中找到 component 對應的那些 DOM element

# Render phase 與 Commit phase

更新畫面時：re-render component

## Render phase

- 再次執行 component function，以新 props 與 state 產生新 React element，以對應新版本畫面
- 比較新版本 React element 和上一次 render phase 產生的舊版本 React element，找出差異處
- 將差異處交給 commit phase 繼續處理

## Commit phase

- 只操作、更新新舊 React element 的差異處對應的實際 DOM element，其餘 DOM element 不動

更新畫面的情況是 React 畫面管理機制的精髓，通常把 React 更新畫面的流程稱為「reconciliation」

# Reconciliation 流程

💡 以一個 Counter component 作為接下來說明的範例：

```
1 import { useState } from 'react';
2
3 export default function Counter() {
4     const [count, setCount] = useState(0); // state 值初始值為 0
5     const handleDecrementButtonClick = () => {
6         setCount(count - 1); //以參數指定新的 state 的值為目前 state 值 -1
7     };
8     const handleIncrementButtonClick = () => {
9         setCount(count + 1); //以參數指定新的 state 的值為目前 state 值 +1
10    };
11    return (
12        <div>
13            <button onClick={handleDecrementButtonClick}>-</button>
14            <span>{count}</span>
15            <button onClick={handleIncrementButtonClick}>+</button>
16        </div>
17    );
18 }
```

# Reconciliation 流程

## 首次 render

- Render phase : state 預設是 `0`，會產生如下的 React element

```
<div>
  <button onClick={handleDecrementButtonClick}>-</button>
  <span>0</span>
  <button onClick={handleIncrementButtonClick}>+</button>
</div>
```

- Commit phase : 將以上的 React element 完整轉換為實際 DOM element，渲染到瀏覽器畫面上

畫面更新則進入 reconciliation 過程

# Reconciliation 流程

## 畫面更新：reconciliation

### 步驟一：呼叫 `setState` 方法更新 state 資料，並發起 re-render

- 呼叫 `setState` 要更新資料時，React 會先以 `Object.is()` 比較要更新的 state 值和舊的值是否相同：
  - 如果相同，判定資料沒有更新，也不需驅動畫面更新，直接中斷後續，不會觸發 re-render
  - 如果不同，判定資料需驅動畫面更新，執行 component function 的 re-render
- 🚧 以上述 Counter component 為例
  - 點擊 increment button 後，會執行 `setCount(0+1)` (因為當前 state 值為 0 )
  - 舊 state 值是 0，新 state 值是 1，React 會以 `Object.is(0,1)` 比較新舊值是否相同
    - → 比較結果為 false，觸發 component function re-render

# Reconciliation 流程

## 畫面更新：reconciliation

### 步驟二：更新 state 資料並 re-render component function

- 根據傳給 `setState` 的新 state 值來更新資料，以新 props 與 state 再次執行 component function
- 💡 以上述 Counter component 為例
  - 以 `setCount(1)` 傳入的新值 `1` 來更新 state 的值，再次執行 Counter 的 component function
  - re-render 呼叫 `useState` 回傳的 count 就是新值 `1`
  - 得到新版的 React element

```
<div>
  <button onClick={handleDecrementButtonClick}>-</button>
  <span>1</span>
  <button onClick={handleIncrementButtonClick}>+</button>
</div>
```

# Reconciliation 流程

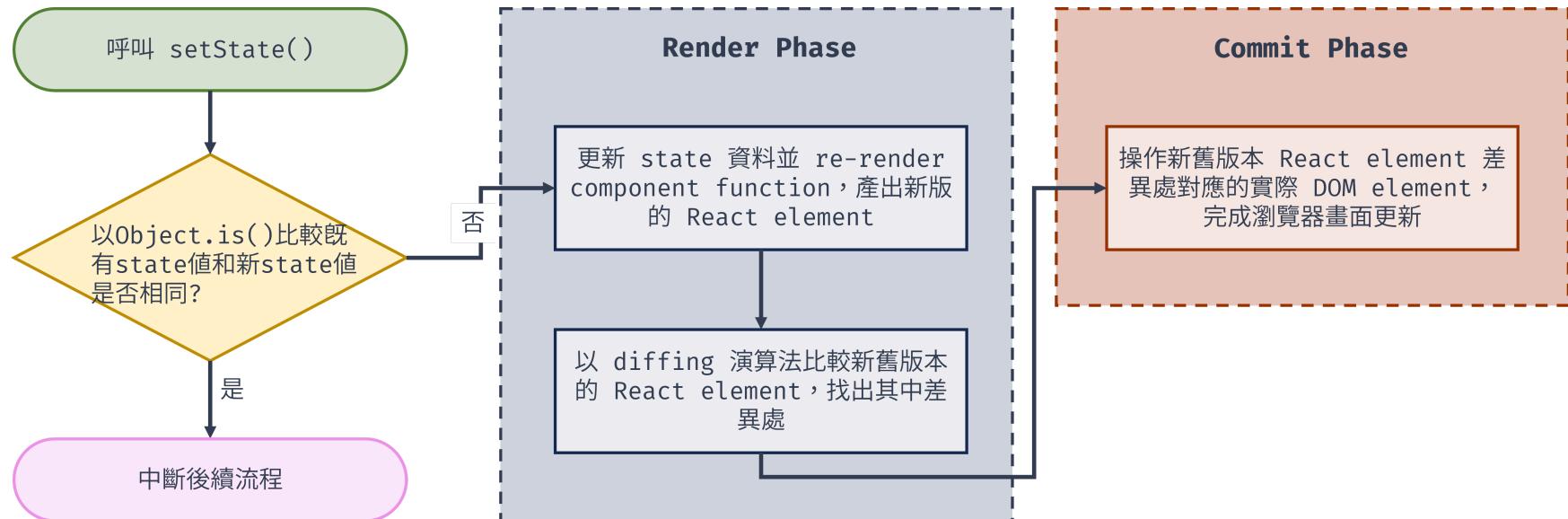
## 畫面更新：reconciliation

### 步驟三：比較新舊版本的 React element，並更新差異處對應的實際 DOM element

- 以 diffing 演算法比較 re-render 的新版 React element 和上一次 render 的舊版 React element，找出兩者差異處
  - 差異處對應的實際 DOM element 是真正要操作的畫面區塊
- 找出差異後，在 commit phase，React 會操作需要被更新的 DOM element，完成畫面更新
  - 其他 DOM element 不動，降低操作 DOM 產生的效能消耗
- 🚧 以上述 Counter component 為例
  - 比較 Counter component 兩次 render 的新舊 React element，找出差異處是 `<span>` 內的文字
  - 在 commit phase，只操作 `<span>` 內的文字

# Reconciliation 流程

## 整體流程示意圖



# setState 觸發的 re-render 會觸發子 component 的 re-render

當 `setState` 觸發 re-render，重新執行 component function 時，如果該 component 內有子 component，也會觸發子 component 的 re-render

- [React child component re-render demo](#)

## component 在兩種情況下會被觸發 re-render

### 1 作為有 state 且呼叫 `setState` 的 component

- component 本身有定義 state，該 state 對應的 `setState` 方法被呼叫時（且要更新的 state 值和既有的值不同）

### 2 作為子 component，被父代以上的 component re-render 影響

- component 沒有因為自己的 `setState` 方法被呼叫而 re-render，而是 component 父代以上的 component 發生 re-render，因而觸發子 component 的 re-render

# Thanks for Listening!

Medium : [React] 認識狀態管理機制 state 與畫面更新機制 reconciliation

# 觀念自我檢測

- React state 的本質是什麼? state 在 React 的畫面管理機制扮演什麼角色?
- State 與 component 的關係是什麼?
- 為什麼 `useState` 的回傳值是一個陣列?
- React 是如何辨認並區分同一個 component 中的多個 state 的?
- 同一個 component 在多個地方被呼叫，它們之間的 state 資料會互通嗎?為什麼?
- 什麼是 render phase 以及 commit phase?
- 解釋 React 更新畫面的 reconciliation 流程
- 一個 component 有哪些可能會被觸發 re-render 的情形?

